**Install Nginx on CentOS 7**

Basic Development Set-up

Disk RAM CPU

15GB 2GB 2

Production Set-up [ Active-Active]

If we want to monitor logs to ensure our security, we must increase our disk size.

Disk RAM CPU

30GB 4GB 4

## Step-1: Add Nginx Repository

To add the CentOS 7 EPEL repository, open terminal and use the following command:

sudo yum install epel-release

## Step-2: Install Nginx

Now that the Nginx repository is installed on your server, install Nginx using the following yum command:

sudo yum install nginx

## Step 3 : Start Nginx

Nginx does not start on its own. To get Nginx running, type:

sudo systemctl start nginx

If you are running a firewall, run the following commands to allow HTTP and HTTPS traffic:

sudo firewall-cmd --permanent --zone=public --add-service=http

sudo firewall-cmd --permanent --zone=public --add-service=https

sudo firewall-cmd --reload

## Changed Parameters

[**worker\_processes**](https://nginx.org/en/docs/ngx_core_module.html#worker_processes)

**worker\_process auto #Default 1**

The number of NGINX worker processes (the default is 1). In most cases, running one worker process per CPU core works well, and we recommend setting this directive to auto to achieve that. There are times when you may want to increase this number, such as when the worker processes must do a lot of disk I/O.

[**worker\_connections**](https://nginx.org/en/docs/ngx_core_module.html#worker_connections) –

**worker\_connections 20000 #Default 1024**

The maximum number of connections that each worker process can handle simultaneously. The default is 512, but most systems have enough resources to support a larger number. The appropriate setting depends on the size of the server and the nature of the traffic and can be discovered through testing.

[**worker\_rlimit\_nofile**](https://stackoverflow.com/questions/37591784/nginx-worker-rlimit-nofile)

The number of simultaneous connections is limited by the number of file descriptors available on the system as each socket will open a file descriptor. If NGINX tries to open more sockets than the available file descriptors, it will lead to the *Too many opened files* message in the error.log.

Check the number of file descriptors using *ulimit*:

**$ ulimit -n**

Now, increase this to a value more than *worker\_process \* worker\_connections*. The value should be increased for the user that runs the worker process. Check the user directive to get the username

## **Epool**

This is an efficient method of processing connections available on Linux 2.6+. The method is like the FreeBSD *kqueue*.

There is also the additional directive *epoll\_events*. This specifies the number of events that NGINX will pass to the kernel. The default value for this is *512*.

events{

use epool;

}

## **multi\_accept**

The multi\_accept flag enables an NGINX worker to accept as many connections as possible when it gets the notification of a new connection. The purpose of this flag is to accept all connections in the listen queue at once. If the directive is disabled, a worker process will accept connections one by one. The following code shows this:

events{

   multi\_accept on;

}

### **tcp\_nopush**

tcp\_nopush is opposite to tcp\_nodelay. Instead of pushing packages as fast as possible, it aims to optimise the amount of data sent simultaneously.  
It will force the package to wait until it gets its maximum size (MSS) before sending it to the client. This directive only works, when sendfile is on.

sendfile on;  
tcp\_nopush on;

It may appear that tcp\_nopush and tcp\_nodelay are mutually exclusive. But if all 3 directives are turned on, nginx will:  
\* ensure packages are full before sending them to the client  
\* for the last packet, tcp\_nopush will be removed, allowing TCP to send it immediately, without the 200ms delay

### **proxy\_ignore\_client\_abort**

proxy\_ignore\_client\_abort on;

|  |
| --- |
| **proxy\_ignore\_client\_abort** on | off; |
| Default: | proxy\_ignore\_client\_abort off; |
|  |  |

Determines whether the connection with a proxied server should be closed when a client closes the connection without waiting for a response.

### **client\_body\_buffer\_size**

client\_body\_buffer\_size 20M;

client\_body\_buffer\_size: This handles the client buffer size, meaning any POST actions sent to Nginx. POST actions are typically form submissions.

### **client\_body\_buffer\_size**

client\_body\_timeout 10;

The client\_body\_timeout directive is responsible for the time a server will wait for a client body or client header to be sent after request. If neither a body or header is sent, the server will issue a 408 error or Request time out.

# reset\_timedout\_connection

reset\_timedout\_connection on;

# When a client connection times out, its associated information may remain in memory depending on the state it was on. Enabling this directive will erase all memory associated with the connection after it times out.

# aio threads

aio threads=default;

It defines a thread pool called default with 32 working threads and a maximum length for the task queue of 65536 tasks. If the task queue is overloaded, NGINX rejects the request and logs this error:

thread pool "*NAME*" queue overflow: *N* tasks waiting

# The error means it is possible that the threads are not able to handle the work as quickly as it is added to the queue. You can try increasing the maximum queue size, but if that does not help, then it indicates that your system is not capable of serving so many requests.

**fastcgi\_send\_timeout**

fastcgi\_send\_timeout 900;

Sets a timeout for transmitting a request to the FastCGI server. The timeout is set only between two successive write operations, not for the transmission of the whole request. If the FastCGI server does not receive anything within this time, the connection is closed.

**fastcgi\_read\_timeout**

fastcgi\_read\_timeout 900;

Defines a timeout for reading a response from the FastCGI server. The timeout is set only between two successive read operations, not for the transmission of the whole response. If the FastCGI server does not transmit anything within this time, the connection is closed.

**keepalive\_requests**

keepalive\_requests 10000;

Defines a timeout for reading a response from the FastCGI server. The timeout is set only between two successive read operations, not for the transmission of the whole response. If the FastCGI server does not transmit anything within this time, the connection is closed.

**Stream**

Stream is used to achieving load balancing

To configure load balancing, create a group of servers, or an upstream group whose traffic will be load balanced. Define one or more [upstream {}](https://nginx.org/en/docs/stream/ngx_stream_upstream_module.html#upstream) configuration blocks in the top‑level [stream {}](https://nginx.org/en/docs/stream/ngx_stream_core_module.html#stream) context and set the name for the upstream group, for example, stream\_backend for TCP servers and dns\_servers for UDP servers:

**stream** {  
    **upstream** docker\_ucp {  
        server [10.100.31.10:443](http://10.100.31.10:443);  
        server [10.100.32.10:443](http://10.100.32.10:443);  
        server [10.100.33.10:443](http://10.100.33.10:443);  
    }  
  
    **upstream** docker\_dtr {  
        server [10.100.10.100:443](http://10.100.10.100:443);  
    }  
  
    **map** $ssl\_preread\_server\_name **$upstream** {  
        hostnames;  
        .ucp.healthvantics.local docker\_ucp;  
        .dtr.healthvantics.local docker\_dtr;  
    }  
  
    **server** {  
        listen 443;  
        listen [::]:443;  
  
        ssl\_preread on;  
        proxy\_pass $upstream;  
    }  
}

Populate the upstream group with upstream servers. Within the [upstream {}](https://nginx.org/en/docs/stream/ngx_stream_upstream_module.html#upstream) block, add a [server](https://nginx.org/en/docs/stream/ngx_stream_upstream_module.html#server) directive for each upstream server, specifying its IP address or hostname (which can resolve to multiple IP addresses) and an obligatory port number. Note that you do not define the protocol for each server, because that is defined for the entire upstream group by the parameter you include on the listen directive in the server block, which you have created [earlier](https://docs.nginx.com/nginx/admin-guide/load-balancer/tcp-udp-load-balancer/#proxy_pass).

**stream** {

**upstream** stream\_backend {

**server** backend1.example.com:12345;

**server** backend2.example.com:12345;

**server** backend3.example.com:12346;

}

The ssl\_preread module can detect more than the protocol. The SNI server name is also supported, which allows for proxy forwarding to different backend servers based on the requested SSL hostname

The ngx\_stream\_ssl\_preread\_module module (1.11.5) allows extracting information from the [ClientHello](http://web.archive.org/web/20180806133249/https:/tools.ietf.org/html/rfc5246#section-7.4.1.2) message without terminating SSL/TLS, for example, the sever name requested through [SNI](http://web.archive.org/web/20180806133249/https:/tools.ietf.org/html/rfc6066#section-3) or protocols advertised in [ALPN](http://web.archive.org/web/20180806133249/https:/tools.ietf.org/html/rfc7301). This module is not built by default, it should be enabled with the --with-stream\_ssl\_preread\_module configuration parameter.

**map** $ssl\_preread\_server\_name **$upstream** {  
        hostnames;  
        .ucp.healthvantics.local docker\_ucp;  
        .dtr.healthvantics.local docker\_dtr;  
    }

**Nginx-Route configuration**

We must include route configuration files for in order to works our routing rule works.

vi /etc/nginx/nginx.conf

Include the routing configuration file location like the below mentioned pattern.

include /etc/nginx/conf.d/config-file-location.conf

and then /etc/nginx/conf.d/config-file-location.conf

sample config

**upstream** ext**-**services {  
        server [10.100.31.11:8080](http://10.100.31.11:8080) max\_fails=0;  
        server [10.100.32.11:8080](http://10.100.32.11:8080) max\_fails=0;  
        server [10.100.33.11:8080](http://10.100.33.11:8080) max\_fails=0;  
        server [10.100.31.13:8080](http://10.100.31.13:8080) max\_fails=0;  
        server [10.100.32.13:8080](http://10.100.32.13:8080) max\_fails=0;  
        }

**server** {  
#       listen       80;  
        listen         8444 ssl;  
       server\_name    \*.[ext-svc.hv-aws.healthvantics.net](http://ext-svc.hv-aws.healthvantics.net);  
        ssl\_certificate    /etc/nginx/certs/ssl/healthvantics.net.pem;  
        ssl\_certificate\_key /etc/nginx/certs/ssl/healthvantics.net.key;  
        root         /usr/share/nginx/html;  
  
        aio threads;  
  
        location / {  
                proxy\_set\_header Host $http\_$host;  
                proxy\_pass          <http://ext-services>;  
                proxy\_http\_version  1.1;  
                proxy\_read\_timeout 900s;  
        }  
  
        error\_page 404 /404.html;  
           location = /40x.html {  
        }  
  
        error\_page 500 502 503 504 /50x.html;  
           location = /50x.html {  
        }  
    }

**proxy\_set\_header Host $http\_$host;**

When NGINX proxies a request, it sends the request to a specified proxied server, fetches the response, and sends it back to the client. It is possible to proxy requests to an HTTP server (another NGINX server or any other server) or a non-HTTP server

Here , In our application

Docker UCP interlock proxy work as a reverse proxy which will redirect the requested URL to upstream servers.

proxy\_set\_header Host $http\_$host;

proxy\_pass          <http://ext-services>;  
                proxy\_http\_version  1.1;  
                proxy\_read\_timeout 900s;